Design of Parallel LDPC Interleaver Architecture: A Bipartite Edge Coloring Approach

Awais Hussein Sani, Philippe Coussy, Cyrille Chavet, Eric Martin

Design of Parallel LDPC Interleaver Architecture: A Bipartite Edge Coloring Approach

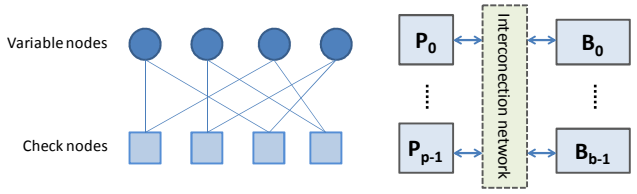
Awais SANI, Philippe COUSSY, Cyrille CHAVET, Eric MARTIN.
Lab-STICC, Université de Bretagne-Sud, Lorient

***Abstract*- Parallel hardware architecture proves to be an excellent compromise between area, cost, flexibility and high throughput in the hardware design of LDPC decoder. However, this type of architecture suffers from memory mapping problem: concurrent read and write accesses to data have to be performed at each time instance without any conflict. In this paper, we present an original approach based on the tanner graph modeling and a modified bipartite edge coloring algorithm to design parallel LDPC interleaver architecture.**

1. INTRODUCTION

Near Shannon limit error correcting capabilities of Low Density Parity Check (LDPC) codes [1] has gained a lot of attention in information theory community. Due to very high decoding throughput and communication performance, LDPC codes are increasingly included in the standards such as DVB-S2 and DVB-T2 [3], WiFi (IEEE 802.11n) [4] or WiMAX (IEEE 802.16e) [5].

LDPC codes are linear block codes and are represented either by parity check matrix H or by Tanner graph [2], which is a bipartite graph. In its tanner graph representation two types of vertices, variable nodes (VNs) and check nodes (CNs), construct the two vertex sets of bipartite graph (cf. Figure 1). VNs represent the codewords (i.e. data to be processed) and CNs corresponds to the parity-check sums (i.e. operations to be done on the data). A VN is connected to a CN by an edge if and only if it is checked by that check node.



a. Tanner graph of an LDPC b. Decoder architecture

Figure 1 LDPC code and architecture

The decoding process is carried out by an iterative message-passing algorithm called "Belief Propagation Algorithm". In this algorithm, VN and CN iteratively exchange their soft-information to qualify the likelihood of the variable in accordance with the associated parity-check equation [1].

Currently, three main families of decoder architecture for LDPC codes have been proposed in the literature:

- Serial decoder
- Partially-Parallel decoder
- Fully-Parallel decoders

Serial decoders suffer from low throughput and fully-parallel decoders from prohibitive area. Thus only partially-parallel architectures are considered in practical hardware design of LDPC decoders. In partially-parallel architecture several processing elements *PE*s are used and set of variable nodes and set of check nodes are allotted to each PE. High throughput requirement can be achieved using a proper number of PEs, while the interconnection network cost tends to be less critical as compared to fully-parallel implementation. Typical architecture for partially-parallel decoder is shown in Figure 1 in which P PEs are connected with B memory banks where $P = B$.

The computation at variable node and check node is quite simple. When designing parallel hardware architecture, the implementation issues mainly arise due to the communication structure between VNs and CNs. The communication structure becomes more and more challenging with the increase in the number of nodes, the number of node degrees, the number of iterations and the parallelism. Hence, parallel implementation suffers from memory accesses collision problem in which more than one PE concurrently accesses the same memory bank to read or write data.

In this paper, we present a memory mapping methodology based on bipartite graph which is able to provide all the PEs conflict free parallel access to the memory banks. This algorithm provides conflict free memory mapping for all types of decoding methods, code types, codeword lengths and code rates.

The remainder of the paper is organized as follows. Section 2 presents a state of the art related to parallel LDPC decoder design. Section 3 introduces the mapping problem. Section 4 describes some definitions related to bipartite graph needed to understand the proposed approach. Section 5 details the mapping algorithm we propose. Finally, section 6 explains the algorithm through a pedagogical example.

2. RELATED WORKS

Currently three classes of approaches to design partially-parallel LDPC decoder architecture exist to tackle the collision problem:

- Design LDPC codes to avoid collision problem [6], [7],
- Use extra memory elements and control logic in the interconnection network in order to remove conflicts [8], [9], [10],
- Find a memory mapping to provide conflict free access to all the memory banks at any time instance [11], [15], [16].

In the first category of decoder implementation, structured or architecture oriented LDPC codes are designed in order to avoid conflicts in accessing data from memory banks. These codes remove the memory access conflicts and simplify the interconnection network through the use of a barrel shifter [6] or a network [7]. However, constraints in the development of structured LDPC codes may cause degradation in code performance.

In the second class of decoder implementation, memory access conflicts are removed either through the addition of extra memory elements or complex interconnection network or both. In [8], configuration memories are used along with 2D-mesh network for LDPC codes of different block size and code rates. In [9], concurrent accesses to the same memory banks are avoided through the use of heterogeneous network. However, this network becomes complex with increasing degree of parallelization and suffers from reduction in the achievable throughput. In [10], Binary de Bruijn network is employed for providing flexible on-chip network for LDPC decoder. Concurrent accesses to the same memory bank are avoided through dedicated routing algorithm which deflects one of the conflicted packets at the router. The flexibility in these complex interconnection networks is paid through additional hardware, increased decoding latency and power consumption.

In the last class, methodologies for solving collision problem are proposed to map the data in different memory banks for conflict free concurrent read/write accesses. In [11], the authors propose to use a mapping algorithm to remove memory conflicts in flexible LDPC decoders. However, the proposed approach is based on a simulated-annealing algorithm, so the user cannot predict when the algorithm will end. Moreover, it fails to optimize either the storage elements or the interconnection network. Finally, different heuristics [15], [16] have been proposed to solve the mapping problem in turbo and LDPC decoding. However, they consider in-place memory access in which data have to be read from and write to the same memory location.

Finally, conflict graph can be used. In this model, a node represents a data and two nodes are connected if and only if the associated data are accessed at the same time. Node coloring approach can then be used to solve the mapping problem: each color corresponds to one memory bank. Unfortunately only one color can be assigned to one node i.e. a data can be stored in only one memory bank. This constraint may require more memory banks than needed (see [17] for more details). Similarly, number of algorithms have been proposed for coloring the edges of a bipartite graph by constructing partitions ([13] and [14] for

example). Unfortunately, like node coloring approaches they can not be used to solve the mapping problem because each data is supposed to be stored in one memory bank only i.e. only one color can be assigned to one edge.

3. PROBLEM FORMULATION

To explain the problem, we consider a set of K data elements $\{d_0, d_1,..., d_{K-1}\}$ and a set of P processing elements $\{PE_0, PE_1,..., PE_{P-1}\}$ which iteratively process these K data elements in N time instances $\{t_0, t_1,..., t_{N-1}\}$.

In order to store these K data elements and to achieve parallel iterative processing of data for high throughput a set of B memory banks $\{b_0, b_1,..., b_{B-1}\}$, where $B = P$, is used. All the memory banks have the same size M which is equal to $M = K/P$.

Mapping problem

Store K data elements in B memory banks in such a manner that P processing elements can access B memory banks in parallel at each time instance for first reading and then writing B data elements without any conflict.

To highlight this problem, we introduce a data access matrix in which we have P rows, related to the processing elements, and N columns, related to the time instances. Data elements in each row are processed by the processing element connected with this row. Similarly data elements in each column need to be accessed in parallel by P processing elements for partially parallel decoding architecture. Figure 2 represent the data access matrix in which we have $K = 6$, $P = B = 3$, $M = 2$ and $N = 6$. Each data is processed by 3 times which shows the iterative nature of the data access. However, data accesses are interleaved in time and there is no regularity in processing the data elements; e.g., data 3 is successively processed in time instances t_1 and t_2 whereas the first access to the data element 4 occurs at time instance t_3.

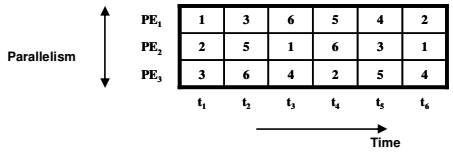


Figure 2: Data Access Matrix

Memory Mapping Constraints

To successfully map the data (i.e. to allow conflict free parallel memory access) in (1) a given number of memory banks and (2) to tackle the iterative nature of data access in error correction coding, the mapping matrix must fulfill the two following constraints:

1- At each time instance, all the memory banks have to be used one and only one time.

2- The bank of the last write access to a data must be the same as the bank of its first read access.

Formal modeling of mapping problem

To tackle the mapping problem, we introduce the concept of multiple read and multiple write access in the formal modeling of mapping problem in which we can not only access the data with in-place strategy (if it is possible) but we can also read a data element from one memory bank and then write it in a different one in order to map the data in minimum required memory banks. This approach is based on the edge coloring of the bipartite graph and presented in section 5.

4. DEFINITIONS

A *graph* $G = (V,E)$ is a collection of node, set V, and edge, set E. If $v,w \in V$ then an edge $(v,w) \in E$ is *incident* to v and to w, and vertices v and w are said *adjacent*. A *subgraph* of G is a graph whose vertices and edges are in G.

To *delete edge* (v,w) from G means to form the subgraph $G - (v,w)$, consisting of all vertices of G and all edges of G except (v,w).

A graph $G = (S_1 \cup S_2, E)$ is *bipartite*, if S_1 and S_2 divide the vertices set so that each edge is incident to a vertex in S_1 and a vertex in S_2 i.e. $S_1 \cap S_2 = \varnothing$.

The *degree* of vertex v is the number of edges incident to v. A graph is *regular* if all vertices have the same degree. A graph is *semi regular,* if either all the vertices in S_1 or all the vertices in S_2 have the same degree.

A *path* P is a sequence of edges $(v_1, v_2), (v_2, v_3),..., (v_{n-1}, v_n)$. The ends of P are vertices v_1 and v_n. If $v_1 \neq v_n$, P is *open*; otherwise P is *closed*. A graph is *connected* if there is a path between any two distinct vertices.

We define a *partition* in semi regular bipartite graph as a subgraph including all time vertices.

Lemma 1: When the degree d_t of the time vertex in a *semi regular* graph is even then we have $d_t/2$ partitions in which each time vertex's degree d_t' is 2.

Lemma 2: When the degree d_t of the time vertex in a *semi regular* graph is odd then we have $\lfloor d_t/2 \rfloor$ partitions in which each time vertex's degree d_t' is 2 and one subgraph in which d_t' is 1.

We finally define a *proper partition* in semi regular bipartite graph as a partition that respects either Lemma 1 or Lemma 2.

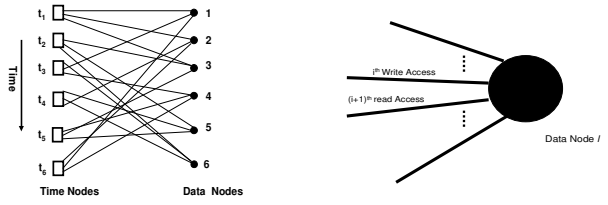
An edge coloring of G is an assignment of a color to each edge in G. An *edge chromatic number*, $\chi`(G)$, is the fewest number of colors necessary to color each edge of a graph so that no two edges incident to the same vertex have the same color.

In [12], König proved that if the maximum vertex degree of a bipartite graph is d then, $\chi`(G) = d$.

5. PROPOSED APPROACH

The proposed algorithm is divided into three steps. In the first step we construct a bipartite graph based on data access matrix. In the second step, we divide our graph into different *proper partitions*. In the third step, the edges of each partition are colored.

Step I: A bipartite graph $G = (T \cup L, E)$ is constructed based on data access matrix (e.g. figure 3) in which vertex set T represents all the time instances and vertex set L represents all the data elements used in the computation. An edge (t, l) is incident to the data element vertex l and to the time instance vertex t if l needs to be processed at t (i.e. data l will be read and next written at time t). Moreover, different data accesses are represented based on the relative position i of edges at the data vertex i.e. first edge at l represents the first read and write accesses and so on. However, the read access that follows the i^{th} write access is the $(i+1) modulo (degree(l))^{th}$ edge at the data node l. An edge that represents the j^{th} read access will be next referred in this paper as a *direct edge* and the edge corresponding to the associated write access as the *induced edge*. This *placement property* will be used during steps II and III. *One interesting property of LDPC decoding is that the number of accesses to data or processing elements at any time instance is always equal which implies that corresponding bipartite graph is always semi regular.* This implies that all the time nodes in the bipartite graph have the same degree d_t.



a. Bipartite graph for **Figure 2** b. l representation

Figure 3: Bipartite representation

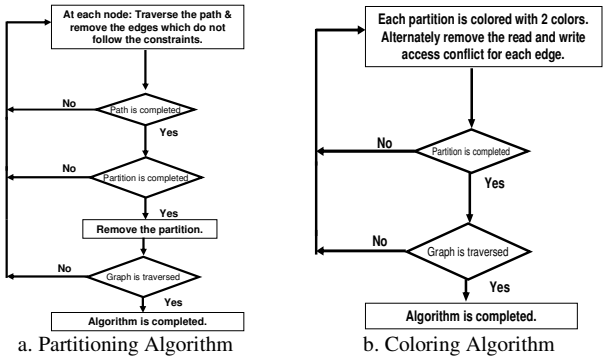
Step II: In this step, bipartite graph G is divided into *proper partitions*. In order to simplify the coloring algorithm next used in step III, one constraint named *partitioning constraint* is introduced: no more than 2 read or write accesses have to be done at each time instance in a proper partition. Following this constraint always allows to construct proper partitions. Each proper partition is constructed using the partitioning algorithm which is shown in Figure 4.a. In this algorithm, two processes working side by side apply at each time and data vertex: *Process of traversal* and *Process of elimination*. Process of traversal randomly selects one edge available at the current data or time node and records its induced edge. *Process of elimination* removes all the edges from the current partition which contradict the partitioning constraint. Hence if d_t' number of selected direct edges (i.e. read accesses) appear in a time node then the remaining (i.e. non-selected) available edges at that time

instance are eliminated. Also, if d_t' number of recorded induced edges (i.e. write accesses) appear in a time node then the direct edges associated to the remaining (i.e. non-recorded) induced edges of that time node are eliminated.

Hence, the algorithm starts constructing a path p_{cur} by choosing any data vertex l_{cur} and then by applying process of traversal which selects randomly an edge (l_{cur}, t_{cur}) to reach at the time vertex t_{cur}. Process of elimination is then applied to remove all the edges which contradict the partitioning constraint. At t_{cur}, the process of traversal is again applied to choose another edge (t_{cur}, l_{next}) to reach at the data vertex l_{next}. Again the process of elimination is applied to remove all the edges which contradict the partitioning constraint. At that time $p_{cur} = \{(l_{cur}, t_{cur}), (t_{cur}, l_{next})\}$. The algorithm continues until p_{cur} is completed, i.e. the process of traversal does not find any valid edge to be included in p_{cur}. The path is added in the current subgraph sg_{cur}. The algorithm tests if the sg_{cur} is a partition (i.e. all the time node has been traversed). Once a partition has been extracted the algorithm stops. Otherwise, the algorithm starts constructing another path p_{next} by using the remaining edges of G (that have not been removed by the process of elimination). Once sg_{cur} becomes a partition, the algorithm starts constructing another partition on the remaining graph $G = G\text{-}sg_{cur}$. Step II is explained through a pedagogical example in the next section.

Step III: Thanks to the construction of proper partitions respecting the partitioning constraint, our coloring algorithm, which flow chart is shown in Figure 4.b, colors each partition with at most two colors. For this it uses a strategy to color each edge in each partition so that there is no conflict in the read and write access at each time node.

For each uncolored partition sg_{cur}, the algorithm starts by removing the read conflict accesses by assigning different color to each edge (l_i, t_{cur}) of t_{cur}. After that, following the *placement property* (see step I description) the algorithm searches in G for each edge (l_i, t_{cur}) of t_{cur} for the induced edge (t_{pred}, l_i). Since only two write accesses are possible at each time node (by partitioning constraint), the algorithm searches in G for the direct edge (l_m, t_k) of the induced edge (t_{pred}, l_m) that belongs to sg_{cur}. The algorithm then colors (l_m, t_k) differently from (l_i, t_{cur}) and continues until it reaches the starting node whose both direct edges are already colored. While the partition is not completely colored the algorithm selects another time node t_{cur} and repeats. It should be noticed that simply giving different colors to both the direct edges at each time node in each partition without taking into account the write access memory conflicts makes the algorithm recursive.



a. Partitioning Algorithm b. Coloring Algorithm

Figure 4: Partitioning and Coloring Algorithms

6. PRACTICAL IMPLEMENTATION

Let us present an example based on the data access matrix in Figure 2. The first step is the construction of bipartite graph which is already depicted in Figure 3. This semi regular bipartite graph has each time vertex with degree d_t is 3. Following Lemma 2, we will have after applying step II, 1 partition in which each time vertex's degree d_t' is 2 and one subgraph in which d_t' is 1. To better understand the modeling approach we propose, we use in his paper a mapping matrix. In this matrix, two columns are added in each time instance column of the data access matrix introduced in section 3. The first column shows the memory banks which are used for read access and second column shows the memory banks which are used for write access at this time instance. The mapping matrix of Figure 2 is shown in Figure 5.

	R	W		R	W		R	W		R	W		R	W		R	W
1			**3**			**6**			**5**			**4**			**2**		
2			**5**			**1**			**6**			**3**			**1**		
3			**6**			**4**			**2**			**5**			**4**		
t_1			**t_2**			**t_3**			**t_4**			**t_5**			**t_6**		

Figure 5: Mapping matrix for data access matrix of Figure 2

The algorithm starts constructing the path p_1 by using the first available edge of data *1* which is (1, t_1), leading to $p_1 = \{(1, t_1)\}$. The selected edge (1, t_1) and its corresponding recorded induced edge (1, t_6) appears respectively as bold and dotted line in Figure 6.a. Using the *placement property* the write access of the edge (1, t_1) indeed appears on the edge (1, t_6). The process of elimination is applied and no edge is removed. The process of traversal continues and adds the edge (t_1, 3) into the path $p_1 = \{(1, t_1), (t_1, 3)\}$. According to the partitioning constraint only two read accesses are possible at each time node. Since two read accesses are completed at t_1 therefore the process of elimination deletes all the remaining edges at t_1: (t_1, 2) in that case. Deleted edges are simply removed from the graph in Figure 6.b. Edge (3, t_5) is then selected and added in the path. Since this edge is both a recorded induced edge and a direct selected edge, it thus appears in bold and dotted line in Figure 6.c.

The process continues until we traverse the path $p_1 = \{(1, t_1), (t_1, 3), (3, t_5), (t_5, 5), (5, t_4)\}$ and reach at the time node t_4. At this point, recorded induced edges at t_2 increase to two and the process of elimination deletes all the direct edges associated to the remaining (i.e. non-recorded) induced edges at t_2. All this process is shown in Figure 6.d.

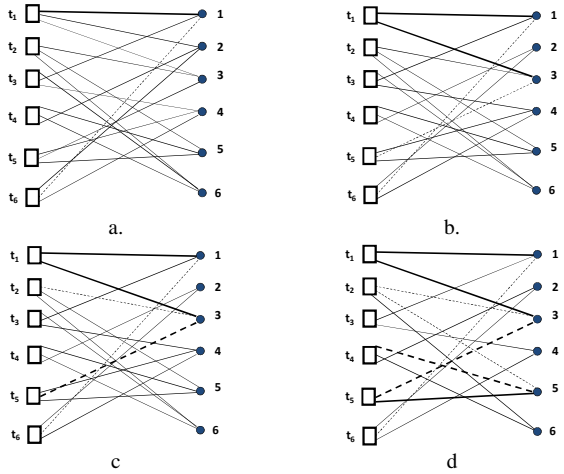


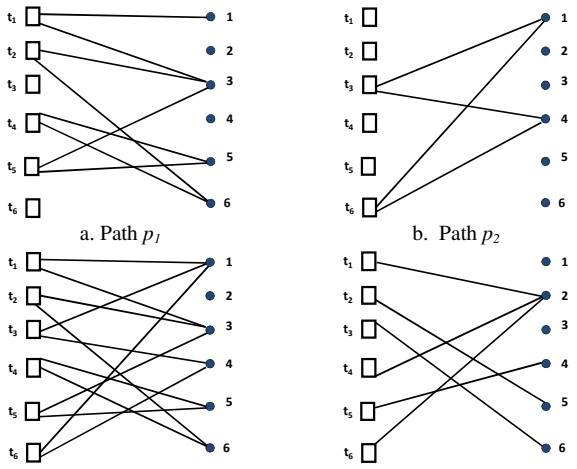
Figure 6: Path construction through Partitioning Algorithm

The traversal continues until the path extends to $p_1 = \{(1, t_1), (t_1, 3), (3, t_5), (t_5, 5), (5, t_4), (t_4, 6), (6, t_2), (t_2, 3)\}$ as shown in Figure 7.a. No more edge can be added in the current path. We thus obtain a subgraph $sg_1 = p_1$. However, the current subgraph sg_1 is not a partition because the time nodes t_3 and t_6 are not included in p_1. Using the process of traversal, the path p_2 is obtained: $p_2 = \{(1, t_3), (t_3, 4), (4, t_6), (t_6, 1)\}$ (see Figure 7.b). The partition sg_1 is the union of all the traversed paths, $sg_1 = p_1 + p_2$ (see Figure 7.c)

Unfortunately, the graph is not completely traversed so the algorithm removes sg_1 to obtain the graph $G' = G - sg_1$ and applies again the processes on the remaining graph to obtain the following paths,

$p'_1 = \{(2, t_1)\}$, $p'_2 = \{(2, t_4)\}$, $p'_3 = \{(2, t_6)\}$, $p'_4 = \{(4, t_5)\}$, $p'_5 = \{(5, t_2)\}$, $p'_6 = \{(6, t_3)\}$. Similarly partition sg_2 is the sum of all the traversed paths as given below, $sg_2 = p'_1 + p'_2 + p'_3 + p'_4 + p'_5 + p'_6$ (see Figure 7.d).

After the construction of sg_2, the algorithm finds that the graph is completely traversed and is terminated.

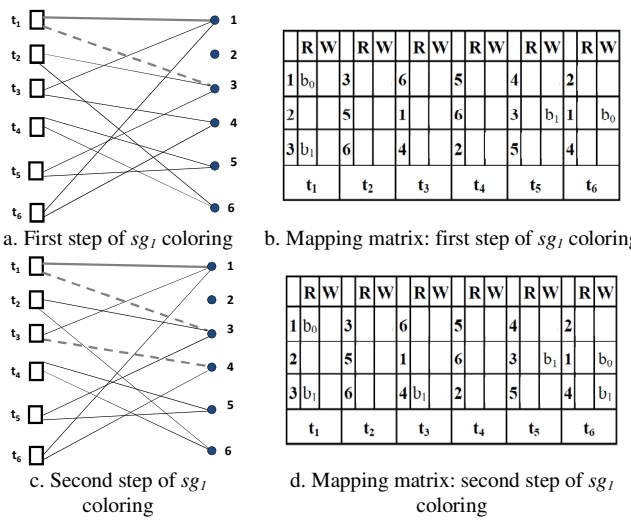


a. Path p_1 b. Path p_2 c. partition sg_1 d. partition sg_2

Figure 7: Path construction through Partitioning Algorithm

After the generation of the partitions, each partition is colored depending on the degree d_t' of its time node. For example, the sg_1 is colored with, d_t' = 2, colors and the sg_2 is colored with, d_t' = 1, color. To color the partition sg_1, we apply the already presented coloring algorithm. We start by coloring the edges connected with t_1 with different colors b_0 and b_1 to avoid a conflict access. Edge $(t_1, 1) = b_0$ and edge $(t_1, 3) = b_1$ as shown in Figure 8.a. In this figure, bold grey straight line represents color b_0 and thin bold grey dotted line represents color b_1. The corresponding mapping matrix is shown in Figure 8.b.

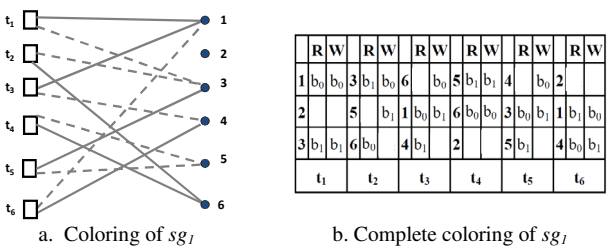
After that we search in G for the induced edges of these previously colored edges. Induced edge of $(t_1, 1)$ is $(1, t_6)$ so we search for the other direct edges that belong to sg_1 and which have an induced edge at t_6 in G. Edge $(t_3, 4)$ must be colored with different color of $(t_1, 1)$ in order to remove the write access conflict at t_6. So we color $(t_3, 4) = b_1$ (see Figure 8.c). The write access of $(t_1, 2)$ occurs also at t_6. However $(t_1, 2)$ does not belong to sg_1, it is not colored at that time. The corresponding mapping matrix is shown in Figure 8.d.



a. First step of sg_1 coloring b. Mapping matrix: first step of sg_1 coloring c. Second step of sg_1 coloring d. Mapping matrix: second step of sg_1 coloring

Figure 8: Conflict free edge coloring of sg_1

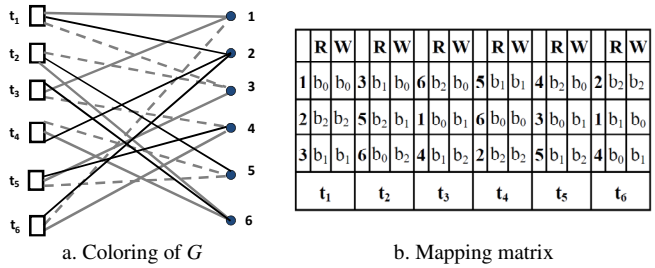
This process continues until the partition is completely colored. The complete coloring of sg_1 is shown in Figure 9.a. The corresponding mapping matrix is presented in Figure 9.b.



a. Coloring of sg_1 b. Complete coloring of sg_1

Figure 9: Conflict free edge coloring of sg_1

The coloring of sg_2 is easier: all the edges are colored with one single color b_2. The complete coloring of G is shown in Figure 10.a. The corresponding mapping matrix is presented in Figure 10.b.



a. Coloring of G b. Mapping matrix

Figure 10: Conflict free edge coloring of G and corresponding mapping

7. CONCLUSION

In this paper, we have presented a conflict free mapping approach for designing any parallel iterative decoding and for any type of LDPC code. The approach introduces the concept of multiple read/write access and uses a modified bipartite edge coloring algorithm. In future works, additional constraints will be added in the algorithm to support the conflict free mapping for specific interconnection networks such as barrel shifter, butterfly or binary De Bruijn. This effort will enhance the design of flexible network of reduced size, higher throughput and lower hardware cost.

REFERENCE

[1] Gallager, R. G., 1962. "Low Density Parity Check Codes". *IRE Trans. Information Theory , 21-28.*

[2] Tanner, R. M., 1981. "A recursive approach to low complexity codes". *IEEE Trans. Inform. Theory, 533-547*

[3] "Frame structure channel coding and modulation for the second generation digital terrestrial television broadcasting system (DVB-T2)," *DVB DocumentA122,* 2008.

[4] IEEE 802.11n. "Wireless LAN Medium Access Control and Physical Layer specifications: Enhancements for Higher Throughput",*IEEE P802.11n/D1.0*, 2006

[5] "Air interface for fixed and mobile broadband wireless access systems," *in P802.16e/D12 Draft*, (Washington, DC, USA), pp. 100-105, IEEE, 2005

[6] M.M. Mansour, N.R. Shanbhag, "High-throughput, LDPC decoders," *IEEE Trans. on Very Large Scale Integration VLSI Systems,* vol.11, pp.976-996, 2003.

[7] Y.Chen, D.Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder". *in Global Telecommunication Conf., 113-117*, 2003.

[8] Theocharides, T., Link, G., Vijaykrishnan, N., and Irwin, M. J., 2005. "Implementing LDPC Decoding on a Network-on-Chip". *in Proc. of the international Conference on VLSI Design, 134-137.*

[9] Kienle, F., Thul, M. J., and When, N., 2003. "Implementation Issues of Scalable LDPC-Decoders". *in Proceeding of 3rd International Symposium on Turbo Codes and Related Topics, Brest, France, 291-294.*

[10] H.Moussa, A.Baghdadi, M.Jezequel."Binary de Bruijn on-chip netwok for a flexible multiprocessor LDPC decoder".*45th ACM/IEEE DAC, p.429-434*, 2008.

[11] F.Quaglio, F.Vacca, C.Castellano, A.Tarable, M G.Asera. "Interconnection Framework for High-Throughput, Flexible LDPC Decoders". *In proceeding Design Automation and Test in Europe Conference and Exhibition*, 2006.

[12] D. König, "Graphok és alkalmazásuk a determinánsok és a halmazok elméletére". *Mathematikai és Természettudományi Értzsitö* 34 (1916) 101-119.

[13] H.N. Gabow, "Using Euler partitions to edge color bipartite multigraphs", *International Journal of Computer and Information Sciences* 5 (1976) 345-355.

[14] R.Cole, J. Hopcroft, "On edge coloring bipartite graphs", *SIAM Journal on Computing* 11 (1982) 540-546.

[15] A.Tarable, S. Benedetto, and G.Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures", IEEE Trans.Inf.Theory, vol. 50, no.9, pp.2002-2009, Sep. 2004.

[16] C. Chavet, P. Coussy, P. Urard and E. Martin, "Static Address Generation Easing: a Design Methodology for Parallel Interleaver Architecture. *In proceeding ICASSP 2010.*

[17] C. Chavet, P. Coussy, "A memory Mapping Approach for Parallel Interleaver design with multiples read and write accesses". In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) 2010.

[18] http://www.ict-davinci-codes.eu/